Week 9

- Abstract Classes
- Interfaces
- Arrays (Assigning , Passing, Returning)
- Multi-dimensional Arrays

Read Pages 342-352

Abstract Classes

- Suppose we have derived Square and Circle subclasses from the superclass Shape.
- We may decide computing area is an operation all subclasses of Shape should provide.
- By making it a superclass operation, we can compute the area of all such objects in a *polymorphic* way as in:

```
Shape [] s = new Shape[5];
double[] areas = new double[5];
s[0] = new Circle(…);
s[1] = new Square(…);
…
for (int i=0; i<5; i++)
     areas[i]  = s[i].computeArea();
```

Abstract method

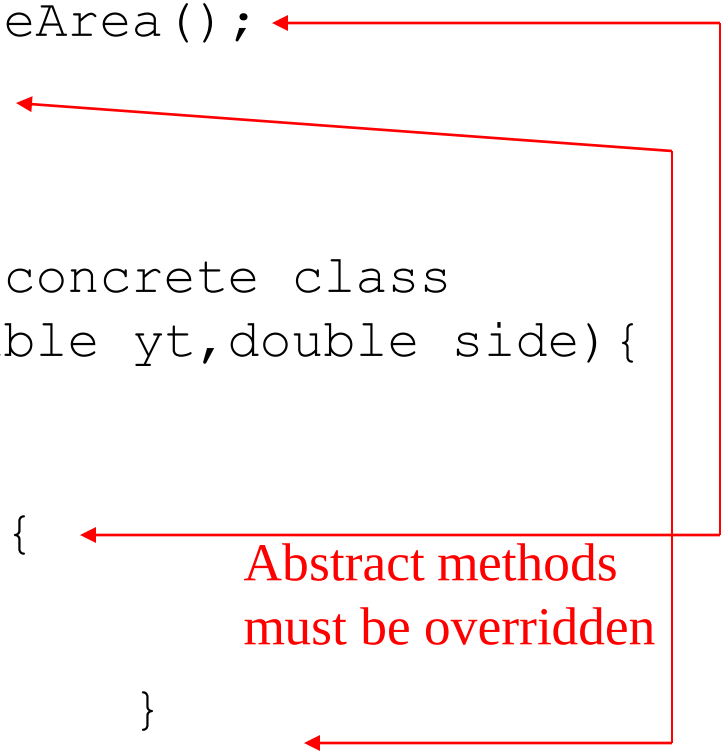
- However, we cannot compute the area of shape, as shape is an abstract entity.
- We can make the computeArea() return a default value of 0.0. However, the subclass may forget to override it resulting in a wrong value
- To force the subclass to override we can declare computeArea() as an abstract method as in:

```
public abstract double computeArea();
```

Abstract vs Concrete class

- Note an *abstract* method has no implementation.
- You cannot construct objects of classes with *abstract* methods - called an *abstract class*.
- In Java, you must declare all abstract classes explicitly with the keyword *abstract*.
- If a subclass of Shape such as Square overrides this method providing an implementation, and has no other *abstract* methods then Square will be considered a *concrete* class.
- Then we can create instances of Square class.

```
abstract class Shape {
      …
    public abstract double computeArea();
    public abstract void print();
    String color;
}
class Square extends Shape{  // concrete class
    public Square(double xt, double yt,double side){
       …
    }
    public double computeArea() {
       return side * side;
    }
    public double print() {   …     }
    …  // other methods and instance variables
}
```




```
Shape aShape;              // OK
aShape = new Shape();      // Error. Cannot instantiate
aShape = new Square(…);    // OK
aShape = null;             // OK
```

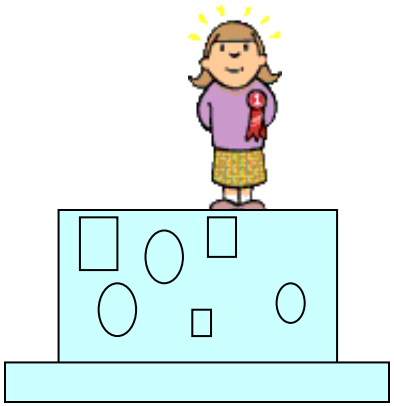
Abstract Class: Some points

- Possible to declare a class abstract even it has no *abstract* methods - preventing users instantiating
- Note that unlike *interfaces*, which we will study next, *abstract classes* can have instance variables.
- Shape may have an instance variable for foreground colour.

A possible scenario

Suppose we have created classes for modelling a square and a circle shape as shown below. Assume these classes are fully tested and used in some existing applications.

```
abstract class Shape {
   public abstract double computeArea();
}
class Circle extends Shape {
      ...
}
class Square extends Shape{
      ...
}
```



Now an applet based application for children requires Square and Circle objects but with the ability to draw these objects on a Graphics object, through a method such as draw(Graphics g).

Design possibilities ...

- We may add a draw(Graphics g) method in the Circle and Square classes.
- As draw() cannot be used in console mode, adding this method makes the class less generic.
- Besides as Circle and Square classes are fully tested it will not be wise to alter these classes.

A better solution is to create an interface (say Drawable) with one method draw(Graphics g) which can be implemented by the Circle and Square derived classes.

What is an interface ?

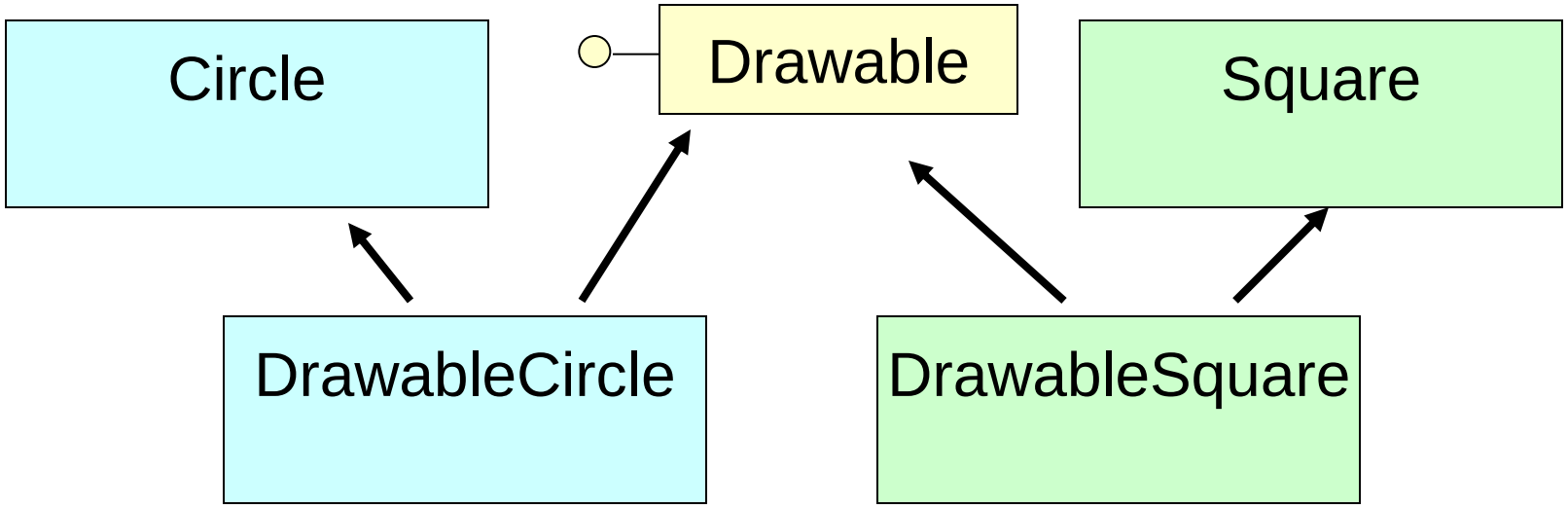
What is an interface?

- A useful construct in Java is an *interface*, which defines a number of *abstract* operations.
- Interface is similar to an abstract class but it is not allowed to have any *instance variables*.
- The interface methods must be defined in the classes implementing the interface.
- In our problem we provide an *interface* Drawable with only one method, as in:

```
interface Drawable{
    public void draw(Graphics g);
}
```

Design using an interface

Now to provide Circle and Square classes which can be drawn on a Graphics object we extend the existing classes making them implement the Drawable *interface*.



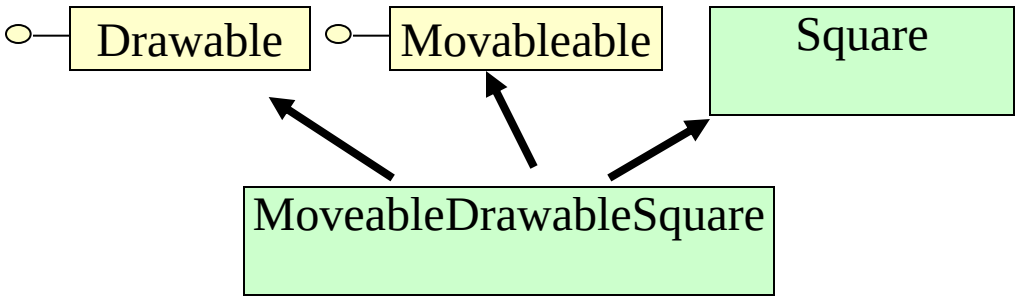
```
class DrawableCircle extends Circle implements Drawable{
   public DrawableCircle(double x, double y, double rad) {
         super(x,y,rad);
   }
   public void draw(Graphics g) {
        g.drawOval((int)(getXCentre()-getRadius() ) ,(int)(getYCentre()-
             getRadius()),  2*(int)getRadius(),     2*(int)getRadius());
   }
}

class DrawableSquare extends Square implements Drawable{
   public DrawableSquare(double xt, double yt, double size) {
        super(xt,yt,size);
   }
   public void draw(Graphics g) {
         g.drawRect((int)getXTop(), (int) getYTop(),
                  (int)getSize(),(int)getSize());
   }
}
```

Java and Multiple Inheritance

- Java does not allow us to extend more than one class.
- If we have a class for student (Student) and employee (Employee) we will not be allowed to create a new class for a student who is both working and studying as in:

```
class EmployedStudent extends Student, Employee {
```

- However we can overcome this to some extent as we can extend one class and implement as many *interfaces* as we need as shown below.



Using an array of interfaces

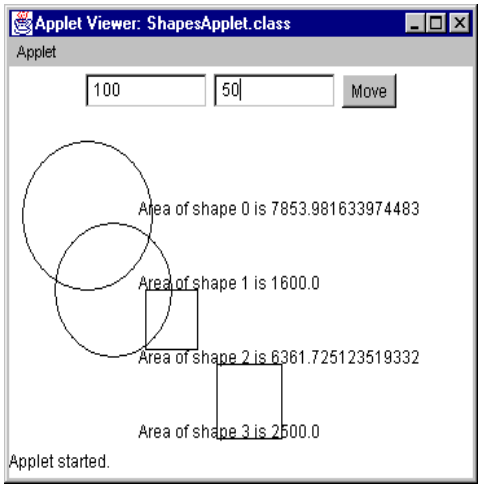
The overriden init() below constructs DrawableCircle and DrawableSquare objects storing them in an array of superclass references before computing their areas.

The paint() method will be called automatically by the system whenever there is a need to call paint the screen

```
public class ShapesApplet extends Applet {
   Shape[] s; // reference to an array of Shape
   int num;   // number of elements
   public void init() {
      num = 5;
      s = new Shape[5];
      s[0] = new DrawableCircle(50,100,100);
      s[1] = new DrawableSquare(75,150,100);
      s[2] = new DrawableCircle(90,150,250);
      s[3] = new DrawableSquare(40,200,100);
      s[4] = new DrawableSquare(40,20,10);
   }
   ...
```

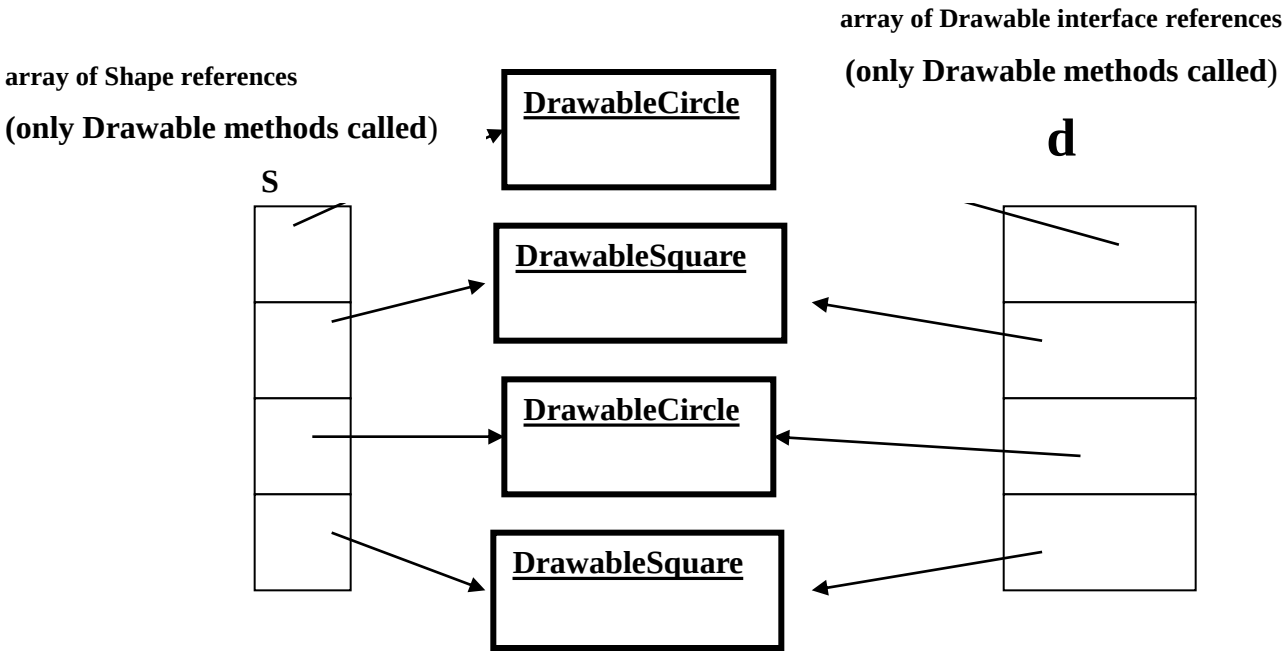
- paint() calls another method drawAllObjects() passing it a reference to Graphics object.
- The paint() method also computes and displays the area of the various shapes in a polymorphic way

```
….
public void paint(Graphics g)
{
      drawAllObjects(g);
      for (int i=0; i<num; i++)
         g.drawString("Area of shape " + i +
           " is " + s[i].computeArea(),100,100+50*i);
}
```



- The method drawAllObjects() below first creates an array of Drawable interface references and sets them to refer to the DrawableSquare and DrawableCircle objects.
- Note that an explicit cast is needed as s is an array of Shape references (though they are referring to Drawable objects).

```
void drawAllObjects(Graphics g)
{
   for (int i=0; i<4; i++)
      ((Drawable)s[i]).draw(g);

}
```


array of Drawable interface references
(only Drawable methods called)
d
array of Shape references
(only Drawable methods called)
S
DrawableCircle
DrawableSquare
DrawableCircle
DrawableSquare

Quiz (Past Exam - MCQ)

1. Which one of the following statements is true?

A) An abstract class must have an abstract method

B) An abstract class cannot be a subclass

C) An abstract class cannot have a constructor

D) An abstract class cannot be instantiated

Quiz (Past Exam - MCQ)

2. Which of the following statement(s) is/are true ?

(I) A class can extend multiple classes

(II) A class can implement multiple interfaces

(A) Both I and II

(B) Only I is true

(C) Only II is true

(D) Both are false

Quiz (Past Exam - MCQ)

3. Which of the following statement (s) is/are true ?

(I) Abstract class cannot have a constructor

(II) An abstract class cannot be instantiated (cannot create an object)

(A) Both I and II are true

(B) Only I is true

(C) Only II is true

(D) Both are false

Abstract Class: Application

- An estate agency has two schemes for selling houses, sale by auction and sale by negotiation.
- For sale by negotiation the charge is:
  sale price * commission rate
- For houses that are auctioned the charge is:
  (actual price – base price) * bonus rate + 2000.
- A program is required to compute charges (polymorphically) and print
- You are required to modify the program - current program only prints the address. (steps needed are outlined in the next slide)

Sold by negotiation

Address	Actual price	Commission rate
34 Kew Crt	340,000	0.05
2 John St	420,000	0.03

Sold by auction

Address	Base price	Actual price	Bonus rate
5 Bet St	560000	565000	0.10
12 Ron Dr	240,000	290,000	0.15

Steps needed

1. In the superclass Sale add a method named double computeCharge()
2. As it cannot be defined (no common scheme) make this method abstract
3. This requires the class also to be abstract
4. Override this method in both subclasses using the scheme specified
5. Now the method computeCharge() can be called polymorphically in the class ManageSales

Abstract Class: Application

```
class Sale
{  private String address;
   public Sale(String add)
   {       address = add;
   }
   public String getAddress()
   { return address;
   }

}
class NegotiatedSale extends Sale
{
   private String address;
   private double commRate;
   private double salePrice;
   public NegotiatedSale(String add,double price,
                                          double cRate )
   {  super(add);
      salePrice = price;
      commRate = cRate;
   }

}
```

Abstract Class: Application

```
class AuctionSale extends Sale
{  private double basePrice;
   private double actualPrice;
   private double bonusRate;
   public AuctionSale(String add, double bPrice, double aPrice,
                                                  double bRate )
   {  super(add);
      basePrice = bPrice;
      actualPrice = aPrice;
      bonusRate = bRate;
   }

}

public class ManageSales
{  public static void main(String args[])
   {  Sale s[] = new Sale[4];
      s[0] = new NegotiatedSale("34 Kew crt",340000,0.05);
      s[1] = new AuctionSale("5 Bet crt",560000, 565000, 0.10);
      s[2] = new AuctionSale("12 Ron dr",240000, 290000, 0.15);
      s[3] = new NegotiatedSale("2 John st",420000,0.03);
      for (int i=0; i<4; i++)
         System.out.println(s[i].getAddress());
   }
}
```

Abstract Class: Application

Abstract Class: Application

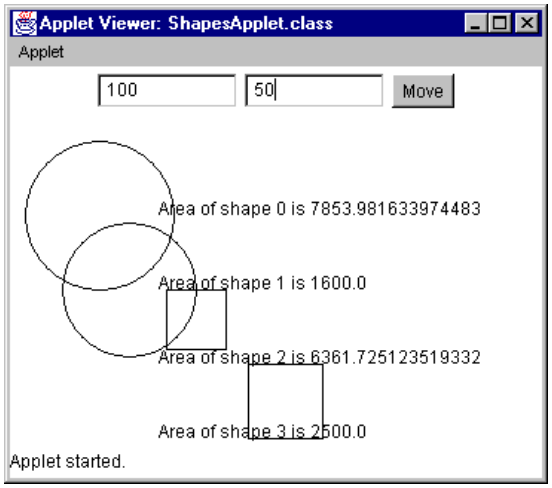
How do we get user input in an applet ?

How do we get user input in an applet?

Using Buttons, TextFields, …

How do I add these to my applet?

Override the init() method in the Applet derived class. Create instances of TextField, Button, … and add them to the applet.



What happens when I press a button?

An event is generated and passed to the event handler for processing

Who can handle Button events ?

An Objects whose class that implements the interface AcionListener which has one method actionPerformed(). This method will process the event (take some action).

How do I register an event handler for button ?

Using the method addAcionListener( ). In this example the same object handles the events - hence - moveButton.addActionListener(this);

Wait till next semester for details

An applet class which takes user input - x and y values

```
import java.applet.*;
import java.awt.event.*;
import java.awt.*;
public class ShapesApplet extends Applet implements ActionListener {
        TextField text1, text2;
        Button moveButton;
        int x, y;
        public void init() {
                text1 = new TextField(10);
                text2 = new TextField(10);
                moveButton = new Button("Move");
                moveButton.addActionListener(this);
                add(text1); add(text2); add(moveButton);
        }
        public void actionPerformed(ActionEvent e) {
            x = Integer.parseInt(text1.getText());
            y = Integer.parseInt(text2.getText());
            repaint();
        }
```

More on Arrays

```
int studentMarks[] = new int[256];
```

Creates a 256 element array of int (initialised to 0).

The size of an array can be specified as above, or at the declaration as follows:

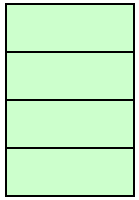
```
int assignmentTotals[] = {6,5, 8, 4, 5};
```

Note that new is not required. Defining and initialising an array can be done at declaration time only.

The array elements can be any type, both primitive and reference. For example, the following is legal:

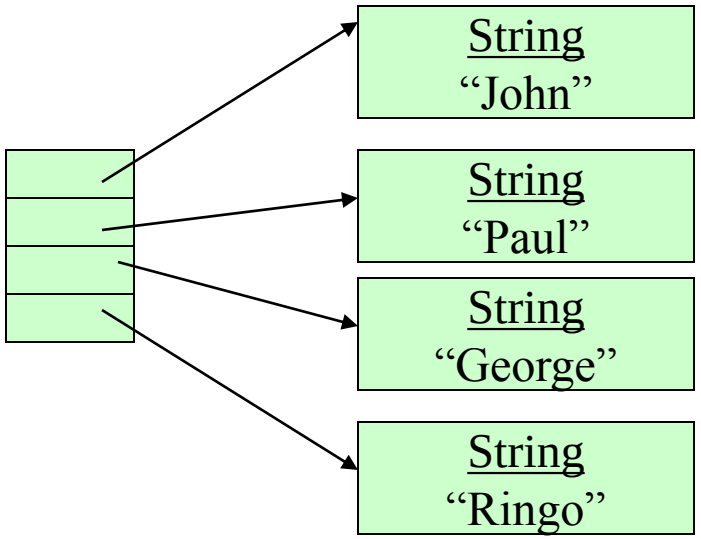
An array of references

```
String fabFour[] = new String[4];
```



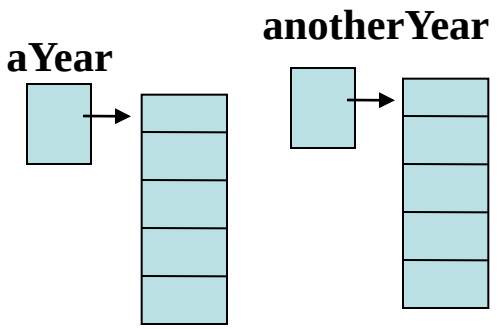
But, of course, these declarations only allocate memory for the references. It is possible to do the declaration and instantiation at the same time:

String fabFour[] = {“John”, “Paul”, “George”, “Ringo”};

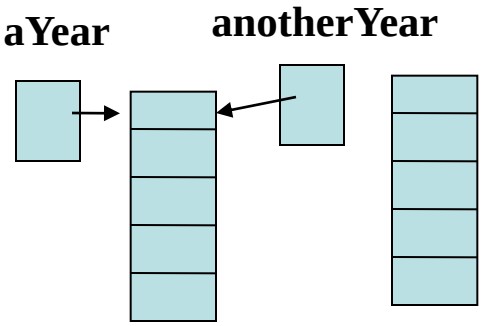


Assigning arrays

As arrays are reference objects, when assigning one to another the reference is copied (not the values).




```
int aYear[] = new int[365];
int anotherYear[] = new int[365];

// setting 2nd array to 1st
anotherYear = aYear;
```

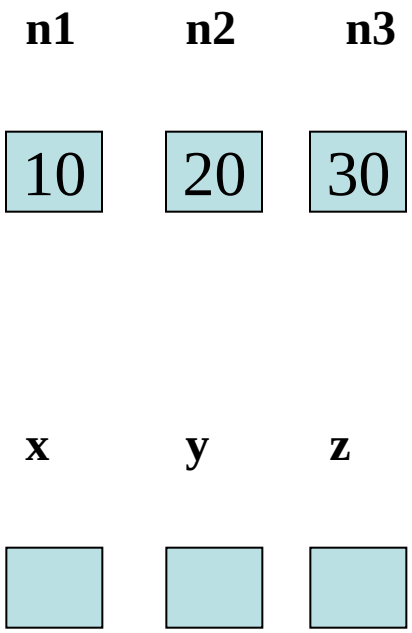


The last statement creates an alias rather than copying the values of aYear onto anotherYear.

If a copy is needed, use: **`java.lang.System.arraycopy()`**

Passing int primitives: What is the output ?

```
public class PassingInts{
  public static void main (String[] args){
    int n1 = 10;
    int n2 = 20;
    int n3 = 30;
    addOne(n1,n2,n3);
    System.out.println("Values are "+ n1 +
                                  ","+n2+","+n3);
   }
  public static void addOne(int x,int y,int z){
     x++;
     y++;
     z++;
  }
}
```



Passing Array of int: What is the output ?

```
public class PassingIntArray{
   public static void main (String[] args){
       int n[] = {10,20,30};
       addOne(n);
       System.out.println("Values are "+
              n[0] + ","+n[1]+","+n[2]);
   }
   public static void addOne(int x[]) {
       x[0]++;
       x[1]++;
       x[2]++;
   }
}
```

n

10	20	30

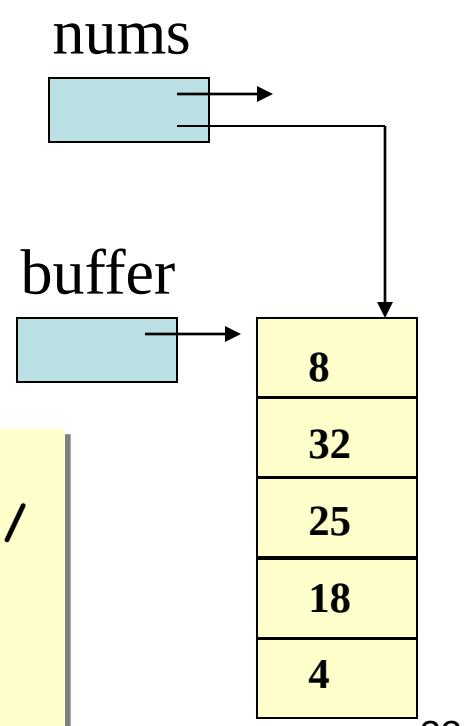
Returning arrays

Since arrays are reference types, it is also possible to return arrays.

What is returned is a reference to the array, so we can store the reference in an appropriate array variable.

For example readMarks() method below sets up an array inside the method and returns the array to the main() method.

```
public static void main(String[] args) {
     int nums[];
     nums  = readMarks(5);
    ...
}
```




```
int[] static readMarks(int numberOfMarks)  {
    int[] buffer = new int[numberOfMarks];  //
    …  // reading the values into the array
    return buffer;
}
```

Multidimensional arrays

Often, a matrix-like data type is needed to solve a problem. Two dimensional array is an array of arrays.

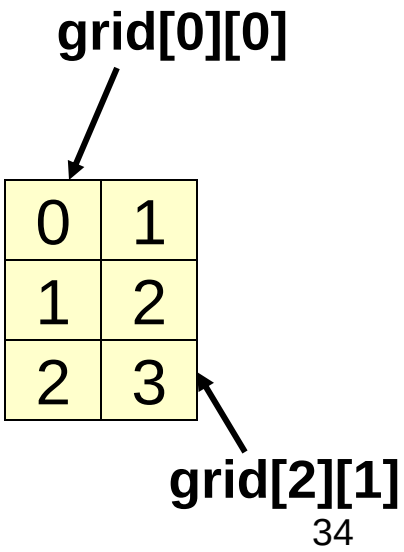
To describe an 8 × 8 grid we can declare int grid[][]; or even better int[] grid[]; or even int[][] grid;

To create an array with 3 rows and 2 columns use

```
int [][] grid = new int[3][2];
```

To set the elements you may use nested loops as in

```
for (int i=0; i<3; i++)
   for (int j=0; j<2; j++)
         grid[i][j] = i+j;
```



Quiz

Write the code to create the 2D arrays and to initialise them as shown using for loops

0	1	2
0	1	2
0	1	2

1	2	2
2	1	2
2	2	1